Sudoku: (Wie) Findet der Computer eine Lösung?

1. Schritt: Sudoku spielen und kennenlernen

	7			2			4	6
	6					8	9	
2			8			7	1	5
	8	4		9	7			
7	1						5	9
			1	3		4	8	
6	9	7			2			8
	5	8					6	
4	3			8			7	

Regeln:

- Sudoku wird auf einem 9x9-Feld gespielt.
- Ziel ist, alle 81 Felder mit den Zahlen 1 bis 9 zu füllen.
- In jeder Zeile darf jede Zahl 1 bis 9 nur einmal vorkommen.
- In jeder Spalte darf jede Zahl 1 bis 9 nur einmal vorkommen.
- In jedem 3x3-Block darf jede Zahl 1 bis 9 nur einmal vorkommen.
- Die vorgegebenen Zahlen dienen als Hinweise; es wird logisch und ohne Raten gelöst.

- Sudoku spielen
- Vorgehen dokumentieren

2. Schritt: Operationalisierung und Modellierung

a) Den allgemeinen Backtracking-Algorithmus auf das konkrete Problem anwenden

- Den Algorithmus *backtrack_all()* auf Sudoku übertragen
- Überlegen, welche Begriffe wie operationalisiert werden können, und den Algorithmus als Pseudocode darstellen
- Mit Stuktog den Backtrack-Algorithmus als Struktogramm darstellen
- Sinnvolle Methoden der Klasse SudokuSpiel entwickeln
- Mit UMLetino ein Implementationsdiagramm für die Klasse SudokuSpiel entwerfen

backtrack_all()	backtrack_sudoku()
FALLS die partielle Lösung eine vollständige Lösung ist merke die partielle Lösung	**FALLS** das Sudoku vollständig ausgefüllt ist merke die vollständige Lösung
Betrachte die nächstmögliche partielle Lösung	Wähle das nächstmögliche leere Feld
DURCHLAUFE alle möglichen Erweiterungen aus dieser partiellen Lösung	**DURCHLAUFE** alle möglichen Zahlen 1 bis 9
FALLS die mögliche Erweiterung zulässig ist Füge die mögliche Erweiterung der partiellen Lösung hinzu backtrack_all() *//partielle Lösung mit möglicher Erweiterung* mache die mögliche Erweiterung wieder rückgängig	**FALLS** die Zahl im Feld zulässig ist Trage die Zahl ein backtrack_all() Mache die Eintragung wieder rückgängig

●: zu operationalisierende Begriffe

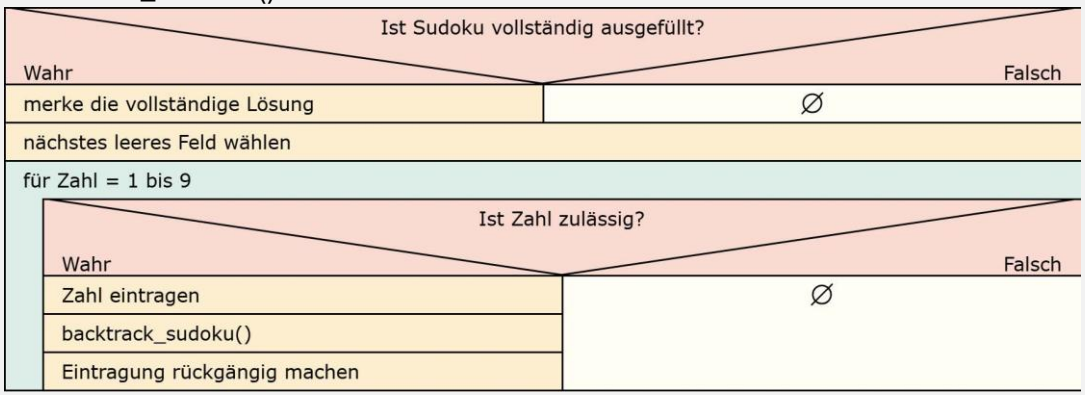
b) Den Sudoku-Algorithmus in Pseudocode reformulieren

```
backtrack_all()

  FALLS das Sudoku vollständig ausgefüllt ist
    merke die vollständige Lösung

  Wähle das nächstmögliche leere Feld

  DURCHLAUFE alle möglichen Zahlen 1 bis 9

    FALLS die Zahl im Feld zulässig ist
      Trage die Zahl ein
      backtrack_all()
      Mache die Eintragung wieder rückgängig
```

```
sudoku_backtracking()

  FALLS kein leeres Feld mehr existiert:
    Lösung merken

  WAEHLE ein leeres Feld (z, s)                    //z: Zeile, s: Spalte

  für jede Zahl x in {1, 2, ..., 9}:
    falls x in (z, s) erlaubt ist:
      setze Feld (z, s) = x
      sudoku_backtracking()
      setze Feld (z, s) wieder auf leer
```

c) benötigte Datenstrukturen, Methoden und Klassen ableiten

backtrack_sudoku()



Methode	Aufgabe / Funktion
SudokuSpiel(int[][] startbelegung)	Konstruktor; übernimmt die Startbelegung und speichert sie intern als aktuelles Sudoku-Feld.
loeseSudoku()	Startet den Lösungsprozess und ruft die rekursive Backtracking-Methode auf. Gibt zurück, ob eine Lösung gefunden wurde.
gibLoesungZurueck()	Liefert die gemerkte vollständige Lösung als 9x9-Feld zurück.
backtrack_sudoku()	Zentrales rekursives Backtracking-Verfahren: prüft Vollständigkeit, wählt ein leeres Feld, testet Zahlen von 1 bis 9, trägt zulässige Zahlen ein, ruft sich rekursiv auf und macht Einträge wieder rückgängig.
istVollstaendigAusgefuellt()	Prüft, ob das Sudoku-Feld keine leeren Felder mehr enthält.
findeNaechstesLeeresFeld()	Sucht das nächste leere Feld im Sudoku, zeilenweise von links oben nach rechts unten.
istZahlZulaessig(int zeile, int spalte, int zuPruefendeZahl)	Prüft, ob eine Zahl an einer bestimmten Position nach Sudoku-Regeln erlaubt ist.
kopiereFeld(int[][] original)	Erzeugt eine Kopie des 9x9-Sudoku-Feldes.

3. Schritt: das Sudoku-Spiel programmieren

- Sudoku mit der Demoversion spielen
- Das SudokuRaster herunterladen und an den drei gekennzeichneten Stellen vervollständigen

Demoversion (gezippte Datei, vor dem Gebrauch entpacken)
https://feg.one/6s0

Raster (gezippte Datei, vor dem Gebrauch entpacken)
https://feg.one/yky

Mögliche Lösung (gezippte Datei, vor dem Gebrauch entpacken)
https://feg.one/umm

4. Schritt: ggf. Lernprogression überprüfen

- Konstruktor der Klasse SudokuSpiel erläutern
- Backtrack-Algorithmus als Struktogramm darstellen
- Im Code die Methode findeNaechstesLeeresFeld() analysieren
- Vertiefung: Code so erweitern, dass ein Rätselheft mit x Sudokus unterschiedlicher Schwierigkeit entsteht (Tipp: Die Schwierigkeit hängt von der Anzahl der vorgegebenen Zahlen ab.)